COMP 4161
NICTA Advanced Course

Advanced Topics in Software Verification

Simon Winwood, Toby Murray, June Andronick, Gerwin Klein

$$\lambda^{\rightarrow}$$

Slide 1

List Homework

- List objects (terms)
- Constructors: cons, nil
- map (that is, map $f\ [x_1, \ldots, x_n] = [f\ x_1, \ldots, f\ x_n]$)
- foldl (that is, foldl $f\ i\ [x_1, \ldots, x_n] = f\ x_1\ (f\ x_2\ (f\ x_3\ (\ldots(f\ x_n\ i)))\ldots)$)

Slide 2

So, what can you do with λ calculus?

λ calculus is very expressive, you can encode:

- logic, set theory
- turing machines, functional programs, etc.

Examples:

$$\begin{array}{lll} \texttt{true} & \equiv \lambda x\ y.\ x & \qquad \texttt{if true}\ x\ y \longrightarrow_\beta^* x \\ \texttt{false} & \equiv \lambda x\ y.\ y & \qquad \texttt{if false}\ x\ y \longrightarrow_\beta^* y \\ \texttt{if} & \equiv \lambda z\ x\ y.\ z\ x\ y & \end{array}$$

Now, `not`, `and`, `or`, etc is easy:

$$\begin{array}{ll} \texttt{not} & \equiv \lambda x.\ \texttt{if}\ x\ \texttt{false}\ \texttt{true} \\ \texttt{and} & \equiv \lambda x\ y.\ \texttt{if}\ x\ y\ \texttt{false} \\ \texttt{or} & \equiv \lambda x\ y.\ \texttt{if}\ x\ \texttt{true}\ y \end{array}$$

Slide 3

More Examples

Encoding natural numbers (Church Numerals)

$$\begin{array}{ll} 0 & \equiv \lambda f\ x.\ x \\ 1 & \equiv \lambda f\ x.\ f\ x \\ 2 & \equiv \lambda f\ x.\ f\ (f\ x) \\ 3 & \equiv \lambda f\ x.\ f\ (f\ (f\ x)) \\ \ldots & \end{array}$$

Numeral n takes arguments f and x, applies f n-times to x.

$$\begin{array}{ll} \texttt{iszero} & \equiv \lambda n.\ n\ (\lambda x.\ \texttt{false})\ \texttt{true} \\ \texttt{succ} & \equiv \lambda n\ f\ x.\ f\ (n\ f\ x) \\ \texttt{add} & \equiv \lambda m\ n.\ \lambda f\ x.\ m\ f\ (n\ f\ x) \end{array}$$

Slide 4

Fix Points

$(\lambda x\ f.\ f\ (x\ x\ f))\ \ (\lambda x\ f.\ f\ (x\ x\ f))\ \ t \longrightarrow_\beta$
$(\lambda f.\ f\ ((\lambda x\ f.\ f\ (x\ x\ f))\ \ (\lambda x\ f.\ f\ (x\ x\ f))\ f))\ \ t \longrightarrow_\beta$
$t\ \ ((\lambda x\ f.\ f\ (x\ x\ f))\ (\lambda x\ f.\ f\ (x\ x\ f))\ t)$

$\mu = (\lambda xf.\ f\ (x\ x\ f))\ (\lambda xf.\ f\ (x\ x\ f))$
$\mu\ t \longrightarrow_\beta t\ (\mu\ t) \longrightarrow_\beta t\ (t\ (\mu\ t)) \longrightarrow_\beta t\ (t\ (t\ (\mu\ t))) \longrightarrow_\beta \ldots$

$(\lambda xf.\ f\ (x\ x\ f))\ (\lambda xf.\ f\ (x\ x\ f))$ is Turing's fix point operator

Slide 5

Nice, but ...

As a mathematical foundation, λ does not work. **It is inconsistent.**

- **Frege** (Predicate Logic, $\sim$ 1879):
  allows arbitrary quantification over predicates
- **Russel** (1901): Paradox $R \equiv \{X | X \notin X\}$
- **Whitehead & Russel** (Principia Mathematica, 1910-1913):
  Fix the problem
- **Church** (1930): λ calculus as logic, `true`, `false`, $\wedge$, ... as λ terms

	with	$\{x \mid P\ x\} \equiv \lambda x.\ P\ x$	$x \in M \equiv M\ x$
Problem:	you can write	$R \equiv \lambda x.\ \texttt{not}\ (x\ x)$	
	and get	$(R\ R) =_\beta \texttt{not}\ (R\ R)$	

Slide 6

We have learned so far...

- λ calculus syntax
- free variables, substitution
- β reduction
- α and η conversion
- β reduction is confluent
- λ calculus is very expressive (turing complete)
- λ calculus is inconsistent

Slide 7

λ calculus is inconsistent

Can find term R such that $R\ R\ =_\beta\ \texttt{not}(R\ R)$

There are more terms that do not make sense:
$1\ 2$, `true false`, etc.

Solution: rule out ill-formed terms by using types.
(Church 1940)

Slide 8

Introducing types

Idea: assign a type to each "sensible" λ term.

Examples:

- for *term t has type α* write $t :: \alpha$
- if x has type α then $\lambda x.\ x$ is a function from α to α
  Write: $(\lambda x.\ x) :: \alpha \Rightarrow a$
- for $s\ t$ to be sensible:
  s must be function
  t must be right type for parameter

  If $s :: \alpha \Rightarrow \beta$ and $t :: \alpha$ then $(s\ t) :: \beta$

Slide 9

THAT'S ABOUT IT

Slide 10

NOW FORMALLY AGAIN

Slide 11

Syntax for $\lambda^{\rightarrow}$

Terms: $t \ ::= \ v \mid c \mid (t\ t) \mid (\lambda x.\ t)$
$v, x \in V, \quad c \in C, \quad V, C$ sets of names

Types: $\tau \ ::= \ \mathsf{b} \mid \nu \mid \tau \Rightarrow \tau$
$\mathsf{b} \in \{\mathtt{bool}, \mathtt{int}, \ldots\}$ base types
$\nu \in \{\alpha, \beta, \ldots\}$ type variables

$$\alpha \Rightarrow \beta \Rightarrow \gamma \quad = \quad \alpha \Rightarrow (\beta \Rightarrow \gamma)$$

Context Γ:

Γ: function from variable and constant names to types.

Term t has type τ in context Γ: $\quad \Gamma \vdash t :: \tau$

Slide 12

Examples

$\Gamma \vdash (\lambda x.\ x) :: \alpha \Rightarrow \alpha$

$[y \leftarrow \texttt{int}] \vdash y :: \texttt{int}$

$[z \leftarrow \texttt{bool}] \vdash (\lambda y.\ y)\ z :: \texttt{bool}$

$[] \vdash \lambda f\ x.\ f\ x :: (\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$

A term t is **well typed** or **type correct**
if there are Γ and τ such that $\Gamma \vdash t :: \tau$

Slide 13

Type Checking Rules

Variables: $$\frac{}{\Gamma \vdash x :: \Gamma(x)}$$

Application: $$\frac{\Gamma \vdash t_1 :: \tau_2 \Rightarrow \tau_1 \quad \Gamma \vdash t_2 :: \tau_2}{\Gamma \vdash (t_1\ t_2) :: \tau_1}$$

Abstraction: $$\frac{\Gamma[x \leftarrow \tau_1] \vdash t :: \tau_2}{\Gamma \vdash (\lambda x.\ t) :: \tau_1 \Rightarrow \tau_2}$$

Slide 14

Example Type Derivation:

$$\frac{\frac{\frac{}{[x \leftarrow \alpha, y \leftarrow \beta] \vdash x :: \alpha}}{[x \leftarrow \alpha] \vdash \lambda y.\ x :: \beta \Rightarrow \alpha}}{[] \vdash \lambda x\ y.\ x :: \alpha \Rightarrow \beta \Rightarrow \alpha}$$

Slide 15

More complex Example

$$\frac{\frac{\frac{\frac{}{\Gamma \vdash f :: \alpha \Rightarrow (\alpha \Rightarrow \beta)} \quad \frac{}{\Gamma \vdash x :: \alpha}}{\Gamma \vdash f\ x :: \alpha \Rightarrow \beta} \qquad \frac{}{\Gamma \vdash x :: \alpha}}{\Gamma \vdash f\ x\ x :: \beta}}{\frac{[f \leftarrow \alpha \Rightarrow \alpha \Rightarrow \beta] \vdash \lambda x.\ f\ x\ x :: \alpha \Rightarrow \beta}{[] \vdash \lambda f\ x.\ f\ x\ x :: (\alpha \Rightarrow \alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta}}$$

$$\Gamma = [f \leftarrow \alpha \Rightarrow \alpha \Rightarrow \beta, x \leftarrow \alpha]$$

Slide 16

More general Types

A term can have more than one type.

Example: $[] \vdash \lambda x.\, x :: \texttt{bool} \Rightarrow \texttt{bool}$

$[] \vdash \lambda x.\, x :: \alpha \Rightarrow \alpha$

Some types are more general than others:

$\tau \lesssim \sigma$ if there is a substitution S such that $\tau = S(\sigma)$

Examples:

$$\texttt{int} \Rightarrow \texttt{bool} \quad \lesssim \quad \alpha \Rightarrow \beta \quad \lesssim \quad \beta \Rightarrow \alpha \quad \not\lesssim \quad \alpha \Rightarrow \alpha$$

Slide 17

Most general Types

Fact: each type correct term has a most general type

Formally:

$$\Gamma \vdash t :: \tau \quad \Longrightarrow \quad \exists \sigma.\, \Gamma \vdash t :: \sigma \wedge (\forall \sigma'.\, \Gamma \vdash t :: \sigma' \Longrightarrow \sigma' \lesssim \sigma)$$

It can be found by executing the typing rules backwards.

- ➜ **type checking:** checking if $\Gamma \vdash t :: \tau$ for given Γ and τ
- ➜ **type inference:** computing Γ and τ such that $\Gamma \vdash t :: \tau$

Type checking and type inference on $\lambda^{\rightarrow}$ are decidable.

Slide 18

What about β reduction?

Definition of β reduction stays the same.

Fact: Well typed terms stay well typed during β reduction

Formally: $\Gamma \vdash s :: \tau \ \wedge \ s \longrightarrow_\beta t \Longrightarrow \Gamma \vdash t :: \tau$

This property is called **subject reduction**

Slide 19

What about termination?

β reduction in $\lambda^{\rightarrow}$ always terminates.

(Alan Turing, 1942)

- ➜ **$=_\beta$ is decidable**
  To decide if $s =_\beta t$, reduce s and t to normal form (always exists, because $\longrightarrow_\beta$ terminates), and compare result.
- ➜ **$=_{\alpha\beta\eta}$ is decidable**
  This is why Isabelle can automatically reduce each term to $\beta\eta$ normal form.

Slide 20

What does this mean for Expressiveness?

Not all computable functions can be expressed in $\lambda^{\rightarrow}$!

How can typed functional languages then be turing complete?

Fact:
Each computable function can be encoded as closed, type correct $\lambda^{\rightarrow}$ term using $Y :: (\tau \Rightarrow \tau) \Rightarrow \tau$ with $Y\ t \longrightarrow_\beta t\ (Y\ t)$ as only constant.

- ➜ Y is called fix point operator
- ➜ used for recursion
- ➜ lose decidability (what does $Y\ (\lambda x. x)$ reduce to?)

Slide 21